# The BL2D Mesh Generator: Beginner's Guide, User's and Programmer's Manual

Patrick Laug, Houman Borouchaki

INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

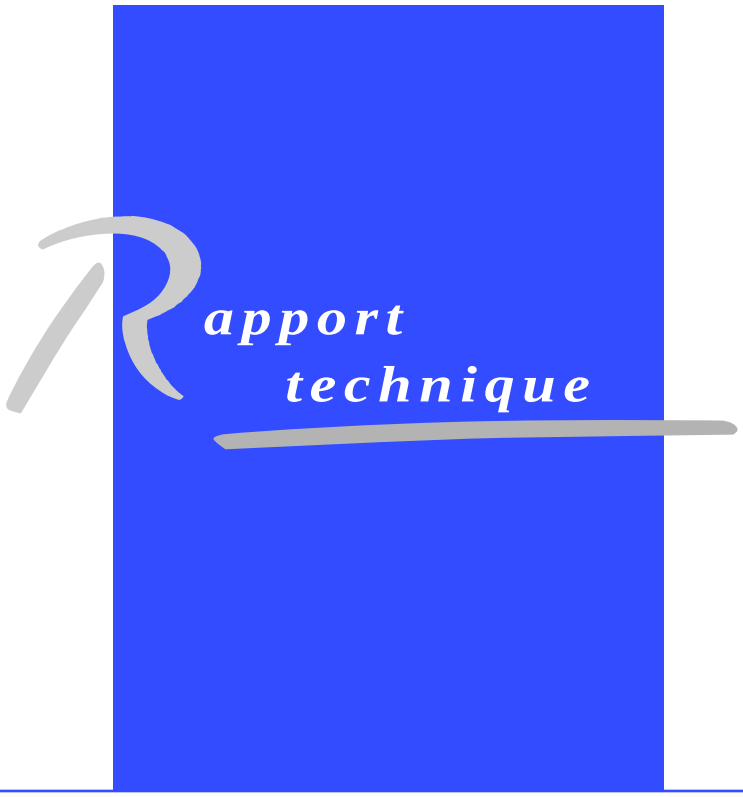
# *The BL2D Mesh Generator: Beginner's Guide, User's and Programmer's Manual*

Patrick LAUG and Houman BOROUCHAKI

**N° 0194**

July 1996

THÈME 4




ISSN 0249-0803

INRIA
ROCQUENCOURT

# The BL2D Mesh Generator: Beginner's Guide, User's and Programmer's Manual

Patrick LAUG* and Houman BOROUCHAKI

Thème 4 — Simulation et optimisation
de systèmes complexes
Projet Gamma

Rapport technique n ° 0194 — July 1996 — 44 pages

**Abstract:** The BL2D software package is a bidimensional, adaptive and anisotropic mesh generator. Its architecture is very modular. This report is made up of three main parts: a beginner's guide (for users who are new to BL2D), a user's manual (for more experienced users) and a programmer's manual (for developers wishing to add new functionalities).

**Key-words:** adaptive mesh, anisotropic mesh, Riemannian metric, data structure.

*(Résumé : tsvp)*

This work was partly supported by the research program "GÉNIE" (DASSAULT AVIATION - INRIA).

The authors wish to thank P.L. George, F. Hecht and E. Saltel for the many stimulating discussions we had with them at INRIA.

* E-mail: Patrick.Laug@inria.fr

Unité de recherche INRIA Rocquencourt
Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
Téléphone : (33 1) 39 63 55 11 – Télécopie : (33 1) 39 63 53 30

# Le mailleur BL2D : guide du débutant, manuel d'utilisation et de programmation

**Résumé :** Le logiciel BL2D est un mailleur bidimensionnel, adaptatif et anisotrope. Son architecture est très modulaire. Ce rapport comprend trois parties principales : un guide du débutant (pour les utilisateurs néophytes), un manuel d'utilisation (pour les utilisateurs confirmés) et un manuel de programmation (pour les développeurs qui voudraient ajouter de nouvelles fonctionnalités).

**Mots-clé :** maillage adaptatif, maillage anisotrope, métrique riemannienne, structure de données.

# Foreword

This report is an augmented and updated translation of a previous version in French [1].

The BL2D software package is available via anonymous ftp from

ftp://ftp.inria.fr/INRIA/Projects/Gamma/bl2d.*

# Preface

The BL2D software package creates 2D isotropic or anisotropic meshes. It can be integrated in an adaptive process. Its software architecture is modular, and two components play a central part:

- The first one discretizes curves in the $\mathbb{R}^2$ space. The methods used for this mesh generator are described in a separate research report [9].
- The second one meshes domains in the $\mathbb{R}^2$ space. This mesh generator is of the generalized Delaunay type [2, 3].

This report is divided into five chapters. Chapter 1 is aimed at users who are new to BL2D. Chapters 2 and 3 are intended for more experienced users. Finally, Chapters 4 and 5 are aimed at developers who want to add new functionalities or to modify some characteristics of BL2D. Below is a brief presentation of these five chapters:

- Chapter 1 is a beginner's guide which introduces the basic tools and gives a complete example of their use.
- Chapter 2 describes the programs that users can execute.
- Chapter 3 describes the formats for input, output or auxiliary files.
- Chapter 4 describes the main data structures that are available to software developers.
- Chapter 5 describes several procedures that are also available to software developers.

# Contents

# Chapter 1

# Beginner's guide

## 1.1 General presentation

The BL2D software package creates 2D isotropic or anisotropic meshes. It can be integrated in an adaptive process. The aim of this chapter is to show how to make use of the different functionalities of BL2D. We present here a typical example of use: the simulation of a physical problem that is modelled by a mathematical formulation, supposed to be known, in a plane domain.

We must first of all define the geometrical and physical data of the domain. The boundary of the domain is defined by straight or curved segments, that are represented by *splines*. We must then mesh the splines and the domain, and make computations using the finite element method. Finally, depending on the results obtained, we must adapt the mesh and make new computations, or else stop the process.

In practice, this comes down to executing several executable programs. Indeed, BL2D has a very modular architecture and is made up of separate programs that communicate with each other via standardized files. A user may thus replace any program by a personal one if it conforms to the standard. The BL2D programs are principally `blsmooth` (compute splines), `blmc` (mesh curves) and `blms` (mesh domain). The main steps to execute are briefly presented below (and are detailed in section 1.2):

1. Define the geometrical and physical data, as well as the data that govern the first mesh. To do so, two initial files must be created:
   - the "G" file, which contains the geometrical and physical data of a plane domain,
   - the "HG" file, which governs the curve mesh and the domain mesh.

   There are several ways of creating these files. The easiest way is to use an interface with an interactive graphics system, for instance the bidimensional mesh and boundary editor EMC$^2$ [11]. It is also possible to use the `blprepro` program which is included in the BL2D software package, but which is not interactive. Finally, the most rudimentary way, that is appropriate only in some cases, simply consists in using a text editor. All the file formats are described in chapter 3.

2. Execute the program that computes "splines" (`blsmooth`). A spline is a straight or curved segment defined by control points. The `blsmooth` program is used to define, as accurately as possible, the domain boundary and if necessary the isolated segments.

3. Mesh the curves (`blmc`) in order to subdivide each spline into a set of edges, that are then transmitted to the domain mesh generator.

4. Mesh the sub-domains (`blprems` and `blms`) in order to subdivide each sub-domain into a set of triangles, and so create the desired mesh.

5. Execute the finite element program. If the mesh must be adapted, create a new file to govern the mesh and go back to step 3. Otherwise, exit from the loop.

## 1.2 Complete example of use

In this section, we present in detail the previous succession of steps. After a few preliminary definitions, we show how to realize an initial isotropic mesh, then adapted anisotropic meshes.

### Definitions and notations

We suppose that the user is working under a UNIX system (other systems may be used, provided some adaptions are made). The files are organized in a *directory* hierarchy.

#### Shell

A *shell* is a command interpreter. To ask the user for typing a command, it outputs a *prompt*, denoted here `%`. Moreover, the *shell* is programmable: a *shell-script* is a program written in the language of the *shell*.

#### Distribution directory

The distribution directory, denoted `~bl2d`, contains programs and data of the BL2D software package. After installation, it contains several subdirectories, amongst which are the following:

- `~bl2d/s`: source programs,
- `~bl2d/`*machine* (*machine* = `alpha`, `hp700`, `ibm`, `sun`, ...): executable programs for a specific target machine (respectively DEC Alpha, HP series 700, IBM RS6000, Sun 4, ...),
- `~bl2d/data`: data that are provided as examples.

It is recommended to add the subdirectory `~bl2d/`*machine* to the environment variable `path` (that tells the *shell* where to search for commands).

#### Working directory

The working directory is any disk space where the user can create files. It is denoted here `~user`, but obviously any other personal directory may be used.

### Creation of an initial isotropic mesh

Let's remember that the necessary data for BL2D can be supplied by an interactive graphics system, for instance EMC$^2$ [11]. However, the initial data are here specified by a text file created by the user (steps 1 and 2 below). Using these input data, an isotropic mesh is obtained (steps 5 to 9 below). These different steps are detailed below:

1. Go to the working directory `~user`, and create there the file `x.0.p` that is printed at the end of section 3.8. This file is normally distributed with the BL2D software package, so just copy it (see the UNIX commands below). Otherwise, it is quite easy to type it again, knowing that it is in free format [5]: the exact number of blanks between two input values is not significant, and any text beginning with the two characters `--` is a comment which is ignored by the program.

```
% cd ~user
% cp ~bl2d/data/quart/x.0.p .
% emacs x.0.p &
```

2. Execute the `blprepro` program (see section 2.9). The input values are the verbosity `1`, the prefix `x` and the iteration `0` (see the dialog below). The program reads the file `x.0.p` and writes two initial files `x.0.g` and `x.0.hg` (figure 1.1):

   - File `x.0.g` (geometrical and physical data) describes a domain whose boundary is made of 5 splines. Splines (1) and (5) are straight segments of length 1, and splines (2) and (4) are straight segments of length 0.5. Spline (3) is defined by the control points 5, 6, 7, 8, 9 in order to approximate a quarter circle. Point 4 is an isolated point inside the domain (this point must be a triangle vertex in all the future domain meshes).
   - The file `x.0.hg` (data to govern the mesh generator) asks for the size $h = 0.1$ at points 4 (isolated), 1, 2, 6, 8, 3 (end points) and 7 (control point inside a spline).

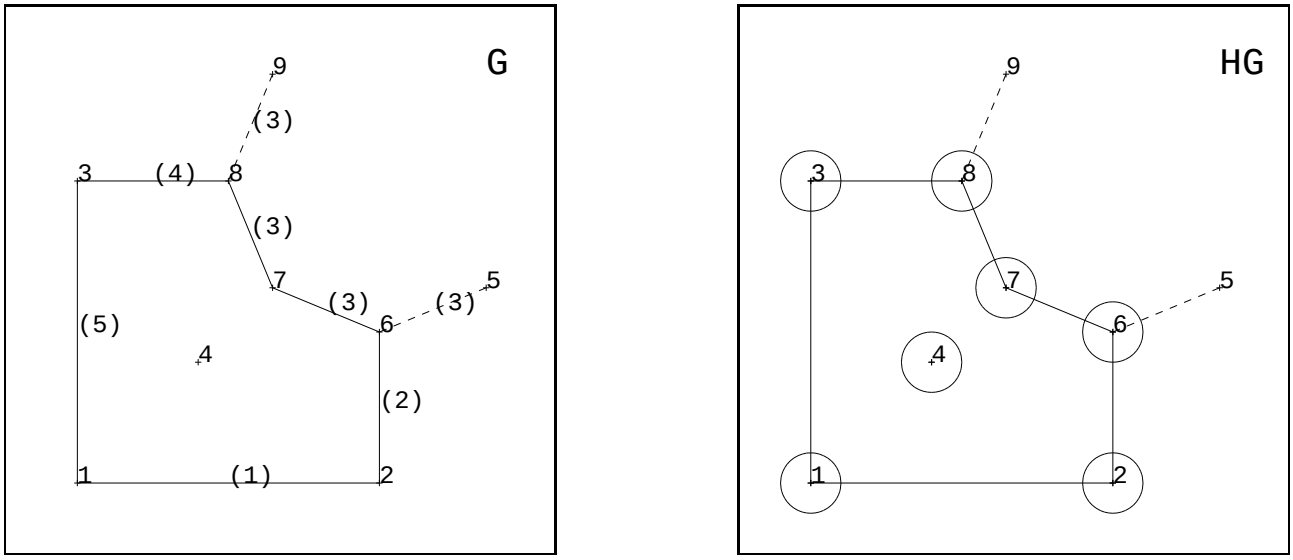



Figure 1.1: Illustration of the initial files `x.0.g` and `x.0.hg`.

```
% blprepro
 ########## blprepro (begin) - create initial files G and HG ##########
 Verbosity?
1
 Prefix?
x
 Iteration?
0
 ---- read  x.0.p

 Isolated points:
 P        4
 Isolated points: total  1

 End points:
 A        1
 B        2
 C        3
```

```
E       6
G       8
End points: total  5
---- write x.0.g
---- write x.0.hg
########## blprepro (end) ##########
```

3. Optional: execute the drawing program `bldraw` (see section 2.2). The inputs are still the verbosity `1`, the prefix `x` and the iteration `0`, and then are driven by menus. The first menu gives a choice for the output graphics terminal. By giving an empty line, one gets the default terminal. By typing then the data below (that are explained in section 2.2), one gets a drawing that illustrates the initial file `x.0.g` (figure 1.1 at the top).

```
% bldraw
 ########## bldraw (begin) - draw files ##########
...
 Files:  g | hg | smooth | c (mc) | hc | s (ms) | hs | h | exp
 Others: b base | z zoom | t test | q quit bldraw
g
 p points | s splines (control polygon) | Return quit draw_g
s
 d draw | n numbers | t tangents | Return quit draw_gs
d
 d draw | n numbers | t tangents | Return quit draw_gs

 p points | s splines (control polygon) | Return quit draw_g

 Files:  g | hg | smooth | c (mc) | hc | s (ms) | hs | h | exp
 Others: b base | z zoom | t test | q quit bldraw
q
 ########## bldraw (end) ##########
```

4. Execute the `blsmooth` program to compute splines (see section 2.10). The inputs are still the verbosity `1`, the prefix `x`, the iteration `0`, and an option number (for instance 2).

5. Execute the `blmc` program to mesh curves (see section 2.6). The inputs are still the verbosity `1`, the prefix `x` and the iteration `0`. The program displays the number of edges created on each spline, and finally the total number of points and edges.

```
% blmc
 ########## blmc (begin) - mesh curves ##########
 ...
 Spline  1  -->  10  edges.
 Spline  2  -->  5  edges.
 Spline  3  -->  8  edges.
 Spline  4  -->  5  edges.
 Spline  5  -->  10  edges.
 MESH -->  39  points,  38  edges.
 ########## blmc (end) ##########
```

6. Execute the `blprems` program (see section 2.8), that creates the input file of the domain mesh generator. The inputs for `blprems` are still the verbosity `1`, the prefix `x` and the iteration `0`.

7. Execute the domain mesh generator `blms` (see section 2.7). The inputs are still the verbosity `1`, the prefix `x` and the iteration `0`. The program displays a lot of statistics concerning the quality of the elements and the time spent.

```
% blms
########## blms (begin) - mesh domain ##########
verbosity <0-few,1-much> --> 1
prefix file --> x
adaption (0,i) --> 0
...
MESH     -->  125 points, 244 triangles
QUALITY  -->  minimum 0.817646, average 0.962795
CPU TIME -->  0.299988 seconds
SPEED    -->  813 triangles / second
########## blms (end) ##########
```

8. The initial isotropic mesh of the domain is obtained. By executing again program `bldraw` (see step 4) with the dialog below, this initial mesh can be visualized (figure 1.2, $i = 0$).

```
% bldraw
bldraw
 ########## bldraw (begin) - draw files ##########
 ...
 Files:  g | hg | smooth | c (mc) | hc | s (ms) | hs | h | exp
 Others: b base | z zoom | t test | q quit bldraw
s
 t triangles (optimized drawing) | tno  triangles (non optimized drawing) ...
t
 ...
 ########## bldraw (end) ##########
```

During the previous steps, BL2D programs create files of different formats (see chapters 2 and 3). The programs try to avoid duplication of data, i.e. each created file is supposed to contain only new data. The more important files created are those containing the curve mesh (format C) and the domain mesh (format MS). One can for instance extract a part of the initial file G and of the new files C and MS to get a file at format AMDBA. The latter gathers some data required by a finite element program (see the `blamdba` program at section 2.1 and the AMDBA format at section 3.1).

### Creation of adapted anisotropic meshes

After the first finite element computation, it is possible to adapt the mesh and to start a new computation, for instance to improve the accuracy of the results. In this case, the results are usually analyzed by a program called *estimator*. The latter creates a file (containing a map of sizes or metrics) that governs the mesh generator in the next iteration. Here, this step is simulated by the program `blestim` (see section 2.4). This program contains several predefined test cases, one of them being defined as

```
case(5)
   theta = atan2(y-1, x-1)
   h1 = 0.4*abs((x-1)**2 + (y-1)**2 - (0.75)**2) + 0.003
   h2 = 0.1
```

The three variables `theta`, `h1` and `h2` define a metric at each point (`x`, `y`). The set of points located at a distance 1 from point (`x`, `y`), in this metric, is an ellipse of inclination `theta` and of sizes `h1` and `h2` along the two main directions (see section 3.4). The expressions of `theta`, `h1` and `h2` are so that, on the circle of center (1,1) and of radius 0.75, we get stretched triangles tangent to this circle, in the ratio of $0.003/0.1 = 3\%$.

The `blestim` program is executed in the following way:

```
% blestim
 ########## blestim (begin) - simulate an estimator ##########
 Verbosity?
0
 Prefix?
x
 Iteration?
0
 Test number?
5
 ########## blestim (end) ##########
```

We can now start the iterations of the adaption loop. There are various ways to execute this loop:

- A first possibility is to execute manually, once again, the programs `blmc`, `blprems` and `blms` (steps 6, 7 and 8 of the previous section), replacing the iteration number 0 by 1. This results in a first adapted mesh, and the loop can continue (with the iteration number 2, 3, ...) until the mesh is considered satisfactory.

- A second possibility, which is generally more convenient, is to use a *shell-script* including control structures. For instance, an adaption loop with three iterations is realized by the following file named `loop.sh`:

  ```
  #!/bin/sh
  for i in 1 2 3
  do (echo x ; echo $i) | blmc
     (echo x ; echo $i) | blprems
     (echo x ; echo $i) | blms
     (echo x ; echo $i ; echo 5) | blestim
  done
  ```

  The first time the file has been created, it is generally necessary to make it executable:

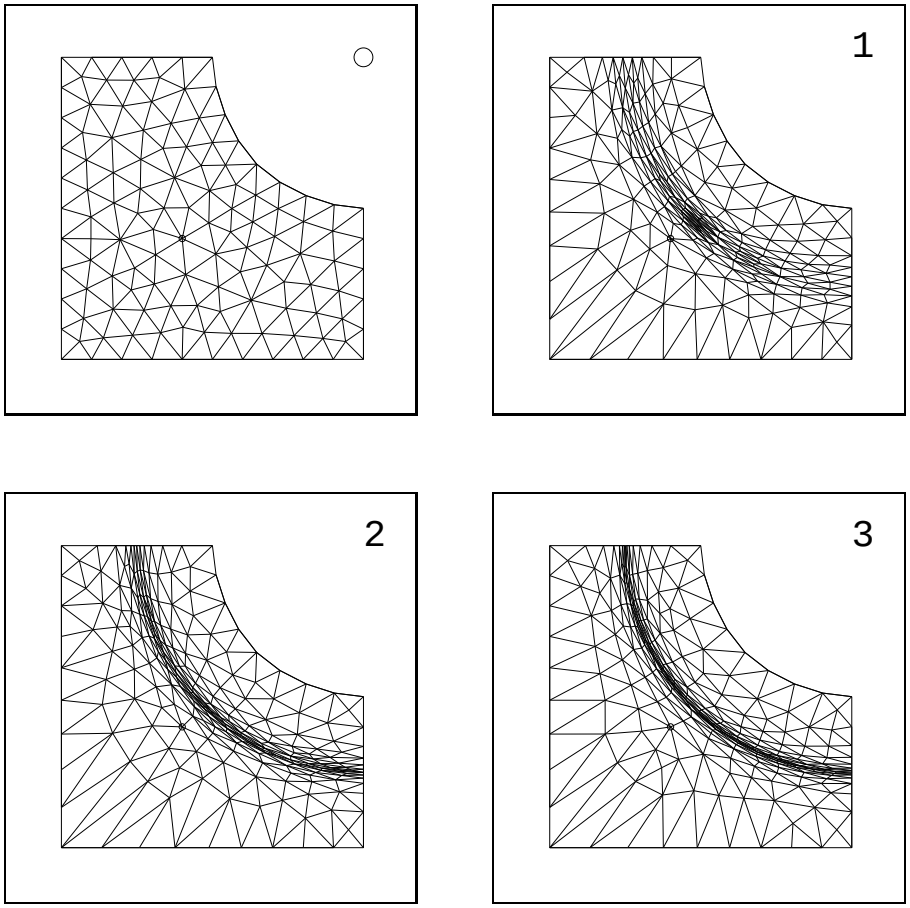
  ```
  % chmod +x loop.sh
  ```

  The loop is then executed by simply typing the name of the *shell-script*:

  ```
  % ./loop.sh
  ```

- A third possibility, which is longer to implement, would be to use a parallel environment like PVM or MPI (with several sub-domains).

Whatever the chosen possibility may be, we get three new meshes (figure 1.2, $i = 1$ to 3). The meshes are more and more conforming to the desired metric. Note that all the meshes respect the isolated point of coordinates (0.4, 0.4). Note also that, in the corners of the domain, the mesh generator automatically subdivides the internal edges whose two end points are on the boundary (this procedure is specific to the finite element method).




∘: isolated point.

Figure 1.2: Initial mesh ($i = 0$) and adapted meshes ($i = 1$ to 3).

## 1.3 File management

During the previous steps, the different BL2D programs transmit each other data through files. The management of these files is almost transparent for the user, thanks to a simple convention on their names that are always of the form:

*prefix.iteration.suffix*

- *prefix* is a name chosen by the user.
- *iteration* is the counter of the adaption loop, whose value is 0 at the creation of the initial mesh (non adapted) and is incremented by 1 at each iteration.
- *suffix* indicates the format of the file (see chapter 3).

# Chapter 2

# Programs

The detailed description of programs that users can execute is given below in alphabetical order. For each program, the following items may be mentioned:

- Standard input (data read from the standard input, which is by default the terminal).
- Input files.
- Output files.

Most of the programs only ask the user for three data values: a *verbosity* value `v` (from 0 = minimum verbosity to 1 = maximum verbosity), a *prefix* `x` and an *iteration* number `i` (see section 1.3). Besides, the name of each file ends with a *suffix* that indicates its format (see chapter 3).

## 2.1 The `blamdba` program

Standard input: *verbosity* v *prefix* x *iteration* i `w`
Input files: x.0.g x.i.c x.i.ms
Output files: x.i.amdba

The `blamdba` program reads three files: x.0.g (geometrical and physical data), x.i.c (curve mesh) and x.i.ms (domain mesh). It extracts parts of these data to create a file at format AMDBA, which is used by several finite element codes (see section 3.1).

The last data `w` in the standard input means that the user wants to write an AMDBA file. In the next future, the program will be able to do the contrary, i.e. read an AMDBA file and create G, C and MS files.

## 2.2 The `bldraw` program

Standard input: driven by menus
Input files: depending on the drawing to be made
Output files: possibly PostScript

The `bldraw` program gives drawings that illustrate different data files. It uses the graphics library Fortran 3D, that allows the user to dynamically choose the output graphics terminal [6, part 3]. It is a menu-driven tool, and the first menu contains two parts called `Files` and `Others`:

```
Files:  g | hg | smooth | c (mc) | hc | s (ms) | hs | h | exp
Others: b base | z zoom | t test | q quit bldraw
```

The `Files` menu offers a selection of one or several input files to be drawn (see formats defined in chapter 3):

- `g`: initial geometrical and physical data (G file),
- `hg`: initial metrics map (HG file),
- `smooth`: splines created by the `blsmooth` program,
- `c` or `mc`: curve mesh created by the `blmc` program,
- `hc`: metrics map created by the `blmc` program,
- `s` or `ms`: domain mesh created by the `blms` program,
- `hs`: metrics map created by the `blms` program,
- `h`: metrics map created by the `blestim` program or by an *estimator*,
- `exp`: parametric expressions of the splines (e.g. to draw tangent vectors).

The `Others` menu is used for other functionalities:

- `b`: basic functions (clear screen, change the characters height or the lines thickness, display a text, ...),
- `z`: zoom the figure,
- `t`: miscellaneous tests,
- `q`: quit the `bldraw` program.

For instance, by typing `g` and replying to the menus that successively appear, an illustration of a G file is obtained (figure 1.1 at the top).

## 2.3 The `blemc2` program

Standard input: *verbosity* v *prefix* x
Input files: x.emc2_bd
Output files: x.0.g x.0.hg

The `blemc2` program creates the initial files G and HG (see section 1.1). It requires the bidimensional mesh and boundary editor EMC$^2$ [11]. Another way to create these initial files is to execute the `blprepro` program (see section 2.9).

## 2.4 The `blestim` program

Standard input: *verbosity* v *prefix* x *iteration* i *case number*
Input files: x.i.c x.i.ms
Output files: x.i.h

The `blestim` program simulates an *estimator*. Usually, an *estimator* analyses the results of a finite element computation and produces from them a sizes or metrics map. Here, the map is analytically defined by the user, in the form of Fortran instructions.

For instance (see section 1.2), the case number 5 of the `blestim` program is reproduced below. On the circle of center (1,1) and of radius 0.75, it defines stretched triangles tangent to this circle, in the ratio of $0.003/0.1 = 3\%$.

```
case(5)
   theta = atan2(y-1, x-1)
   h1 = 0.4*abs((x-1)**2 + (y-1)**2 - (0.75)**2) + 0.003
   h2 = 0.1
```

It is easy to add new cases in the source program `blestim.f90` that is included in the distribution library (see section 1.2).

## 2.5 The `blinterpol` program

Standard input: *verbosity* v *prefix* x *iteration* i
Input files: x.i.bb x.i+1.is
Output files: x.i.bbi

The `blinterpol` program uses two input files that respectively contain:

- the solution of a finite element computation on a mesh $i$,
- the barycentric coordinates of the points of mesh $i+1$ with respect to mesh $i$ (see section 3.5).

It outputs an interpolated solution that can be used to initialize the next computation (the distributed version makes an interpolation of type P1 but can be modified).

## 2.6 The `blmc` program

Standard input: *verbosity* v *prefix* x *iteration* i
Input files: if $i = 0$: x.0.g x.0.smooth x.0.hg
if $i \geq 1$: x.0.g x.0.smooth x.i-1.c x.i-1.h
Output files: if $i = 0$: x.0.c x.0.hc
if $i \geq 1$: x.i.c x.i.hc x.i.ic

The `blmc` program is a curve mesh generator: it discretizes each curved segment into edges. The sizes of these edges are determined by the input file `x.0.hg` if $i = 0$, and `x.i-1.h` if $i \geq 1$.

## 2.7 The `blms` program

Standard input: *verbosity* v *prefix* x *iteration* i
Input files: if $i = 0$: x.0.mc x.0.hc
if $i \geq 1$: x.i.mc x.i.hc x.i-1.ms x.i-1.h
Output files: if $i = 0$: x.0.ms x.0.hs
if $i \geq 1$: x.i.ms x.i.hs x.i.is

The `blms` program is a domain mesh generator: it discretizes each sub-domain into triangles. The sizes and the shapes of these triangles are determined by the input file `x.i.hc` created by the curve mesh generator.

## 2.8 The `blprems` program

Standard input: *verbosity* v *prefix* x *iteration* i
Input files: x.0.g x.0.smooth x.i.c
Output files: x.i.mc

The `blprems` program creates the input file of the `blms` domain mesh generator. These data are in the MC format (see section 3.6).

## 2.9 The `blprepro` program

Standard input: *verbosity* v *prefix* x *iteration* i=0
Input files: x.0.p
Output files: x.0.g x.0.hg

The `blprepro` program creates the initial files G and HG (see section 1.1). It reads an input file created by the user (see section 3.8). Another way to create these initial files is to execute the `blemc2` program (see section 2.3).

## 2.10 The `blsmooth` program

Standard input: *verbosity* v *prefix* x *iteration* i=0 *option number*
Input files: x.0.g
Output files: x.0.smooth

The `blsmooth` program executes three different steps:

1. It checks the initial files G and HG: the points coordinates must be different each other, the isolated points and the end points must be different each other, sizes or metrics must be specified at the isolated points and at the end points, ...
2. It computes the parametric expressions of each spline,
3. It computes the polygonal segment and creates a SMOOTH file (see section 3.9).

The *option number* selects the interpolation method: pure Catmull-Rom (not recommended), modified Catmull-Rom, elastic curve with minimum energy (see the G input format, section 3.3).

# Chapter 3

# Formats

### Main goals

A scientific computation code is generally made up of several software components that communicate each other through files. These files must conform to a certain data organization or "format". An analysis of existing formats lead us to specify a new organization of information [7]. More precisely, the desired possibilities are listed below (some of them may be contradictory):

- Definition of simple structures, both for creating and for handling them.
- Presence of all information that is useful to any kind of scientific problem.
- Uniqueness (i.e. no duplication) of information.
- Evolutivity in the organization of information.
- Efficiency with regard to time and memory space.
- 2D or 3D space.
- Straight or curved P2 elements (position of nodes that are not vertices).
- Possibility to handle all the already known elements, including those recently studied (e.g. *mortar* elements) and to handle in the future types that are unknown today.
- Cracks.
- Tangents - normals.
- Adaption, requiring in particular a remeshing of the boundaries.
- Independence towards the CAD systems (that are diverse and changing).
- Possibility to handle in a simple way the case of only one type of element.
- Particular structures (e.g. grid).
- Pasting two incompatible meshes together.
- Free boundary meshes.
- Multi-physical problems.

### Presentation of the formats

In the context of the bidimensional BL2D mesh generator, we defined the formats hereafter, that are classified into three categories:

1. The input formats of BL2D:
   - G: geometrical and physical data,
   - H: sizes or metrics map.
2. The output formats of BL2D (in fact, in the case of a mesh adaption loop, these output formats become input formats at the next iteration):
   - C: curve mesh,
   - MS: domain mesh.

3. The auxiliary formats. It concerns formats that are interfaces to other software packages (AMDBA and P) or internal to BL2D (IS, MC and SMOOTH):
   - AMDBA: domain mesh with physical data,
   - IS: barycentric coordinates, with a view to interpolation,
   - MC: curve mesh used as an input of the domain mesh generator,
   - P: input of the `blprepro` preprocessor,
   - SMOOTH: splines representation.

This organization with several formats attempts to reach the preceding goals as well as possible. Notably, the distinction between the geometrical and physical data, the curve mesh and the domain mesh, avoids a duplication of information in separate files. This results in gains for simplicity, disk space and input/output time.

**Common remarks to all the formats**

1. These formats define, at user's choice, text files (*ascii*) or binary files. The text files make the communication easier between programs that are written in different languages or compiled on different machines. The binary files avoid coding and decoding time.

2. Each floating-point number must be readable into a double-precision variable in the Fortran 90 or C language. The number of significant digits is depending on the machine and the used programs.

   – Read example in Fortran 90:
   ```
   double precision d
   read (file,*) d
   ```

   – Read example in C:
   ```
   double d;
   scanf(file, "%lf", d);
   ```

3. The formats generally begin with a line of the form:

   *format* `v1`

   - *format* is the format name, that can be checked when reading.
   - `v1` is the version number, in order to facilitate the future software evolution.

4. For convenience sake, these formats are presented below in alphabetical order.

## 3.1 The AMDBA auxiliary format

### Purpose

The purpose of the AMDBA format is to describe a domain mesh and its physical references. Il is used by several finite element codes. However, it is less general than the G, C and MS formats put together. In particular, it does not contain the physical references of the edges or information to compute the coordinates of P2 nodes.

These data are generated by the `blamdba` program.

### Definition of the AMDBA format

NP NT
For all the points ..... $i = 1..NP$
$i$ $x_i$ $y_i$ $\varphi_i$
For all the triangles ..... $i = 1..NT$
$i$ $p_{i1}$ $p_{i2}$ $p_{i3}$ $\varphi_i$

### Notations

NP Number of points.

NT Number of triangles.

For a point $i = 1..NP$:

$x_i$ $y_i$ Coordinates of the point.

$\varphi_i$ Physical reference of the point.

For a triangle $i = 1..NT$:

$p_{i1}$ $p_{i2}$ $p_{i3}$ Numbers of the three vertices given counterclockwise.

$\varphi_i$ Physical reference of the triangle.

### Example of a file in the AMDBA format

The AMDBA file below has been used in our example (see section 1.2).

```
121   202
1   0.4000000   0.4000000   10
2   0.0000000   0.0000000   20
3   1.0000000   0.0000000   30
...
1   69   70   98   200
2   96   92   93   200
3   35   58   57   200
...
```

## 3.2 The C output format

### Purpose

Let's remember that, in the G format, a domain description involves splines (that are straight or curved). Each spline can then be meshed, i.e. subdivided into edges. The purpose of the C format is to describe this set of edges.

These data are generated by the `blmc` program.

### Preliminary definitions

For a given curve mesh, many types of interpolation are possible. If the mesh is of type P1, the nodes are coinciding with the ends of the edges. If the mesh is of type P2, each edge has a supplementary node between its two ends. In our example (figure 3.1), the mesh contains 3 edges ($a_1$, $a_2$, $a_3$), 4 end points ($n_1$, $n_3$, $n_5$, $n_7$) and 7 nodes ($n_1$, $n_2$, ..., $n_7$). In order to allow any type of interpolation, the C format contains P1 nodes only, but associates to each node its curvilinear abscissa $s$ on the considered spline ($0 \leq s \leq$ length of the spline). Therefore, it is possible to compute the coordinates of the other nodes.

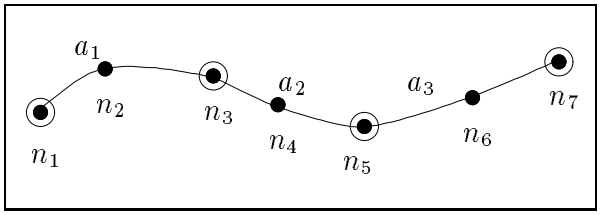



Figure 3.1: Mesh of a curve with P2 nodes.

### Definition of the C format

`C v1`
NP NPI NPE NPC NS
For all the points ..... $i = 1..\text{NP}$
$x_i$ $y_i$
For all the splines ..... $i = 1..\text{NS}$
$np_i$
$(p_{ij} \quad s_{ij})_{\ j=1..np_i}$

### Notations

| | |
|---|---|
| NP | Total number of points: NP = NPI + NPE + NPC. |
| NPI | Number of isolated points. |
| NPE | Number of end points. |
| NPC | Number of remaining points (i.e. neither isolated nor end points). |
| NS | Number of splines. |

For a point $i = 1..\text{NP}$:

| | |
|---|---|
| $x_i$ $y_i$ | Coordinates of the point. |

For a spline $i = 1..\text{NS}$:

| | |
|---|---|
| $np_i$ | Number of points on the spline, end points included ($np_i \geq 2$). |
| $p_{ij}$ | Numbers of the points on the spline ($1 \leq p_{ij} \leq \text{NP}$), given in the following order: |
| $p_{i,1}$ | Number of the first end point. |
| $p_{i,2..np_i-1}$ | Numbers of the points that are internal to the spline, starting from the first end point and going as far as the second end point. |
| $p_{i,np_i}$ | Number of the second end point. |
| $s_{ij}$ | Curvilinear abscissa of the point $p_{ij}$. For $j$ varying from 1 to $np_i$, $s$ increases from 0 to the length of the spline. |

### Remarks

When the points are numbered (from 1 to NP), we consider successively:

- the NPI isolated points defined in G,
- the NPE end points defined in G,
- the NPC points created by the curve mesh generator.

The total number of edges is NPC + NS.

The physical references of points and edges, that are already described in the G format, are not duplicated in the C format. They can be obtained by adequate functions (see section 5.2).

### Example of a file in the C format

The C file below has been used in our example (see section 1.2).

```
C v1
39 1 5 33 5
  0.4000000000000000   0.4000000000000000
  0.0000000000000000   0.0000000000000000
  1.0000000000000000   0.0000000000000000
  ...
  0.9032169485291625   0.5120700124670046
  0.8102791308792039   0.5419733047033631
  0.7234173490701071   0.5865454450721955
  0.6464466094067263   0.6464466094067263
  0.5865454450721955   0.7234173490701072
  0.5419733047033631   0.8102791308792039
  0.5120700124670046   0.9032169485291626
  ...
11
 2   0.0    7   0.1    8   0.2    9   0.3   10   0.4   11   0.5
12   0.6   13   0.7   14   0.8   15   0.9    3   1.0
 6
 3   0.0   16   0.1   17   0.2   18   0.3   19   0.4    5   0.5
 9
 5   0.0000000000000000
20   9.7739365406961959E-02
21   0.1954787308139239
22   0.2932180962208858
23   0.3909574616278478
24   0.4886968270348097
25   0.5864361924417716
26   0.6841755578487336
 6   0.7819149232556956
 6
 6   0.0   27   0.1   28   0.2   29   0.3   30   0.4    4   0.5
11
 4   0.0   31   0.1   32   0.2   33   0.3   34   0.4   35   0.5
36   0.6   37   0.7   38   0.8   39   0.9    2   1.0
```

## 3.3 The G input format

### Purpose

The purpose of the G format is to describe the geometrical and physical data of a plane domain. These data are normally generated by an interactive graphics system or by a preprocessor (see chapter 1).

### Preliminary definitions

#### Recall

A domain can be made up of one or several sub-domain(s). Each sub-domain is delimited by its boundary.

To execute finite element computations, it is necessary to mesh the domain. Several meshes can be created on a same domain (for instance to adapt the mesh and get greater accuracy, or to solve a multi-physical problem). Any mesh created must respect the boundaries of the sub-domains. The user may also wish that each mesh respects some given segments or points (we say then that we have isolated segments or isolated points). A segment may have only zero or one end point common to a boundary segment (if it had two common end points, it would be a boundary segment itself).

For instance (figure 3.2), the domain $\Omega$ is composed of two sub-domains $\Omega_1$ and $\Omega_2$. The boundary of sub-domain $\Omega_1$ is the connected component $\Gamma_1 \cup \Gamma_2$. The boundary of sub-domain $\Omega_2$ is the two connected components $\Gamma_2 \cup \Gamma_3$ and $\Gamma_4$ (which delimits a "hole"). $\Omega_2$ contains the isolated segment $S$ and the isolated point $P$.

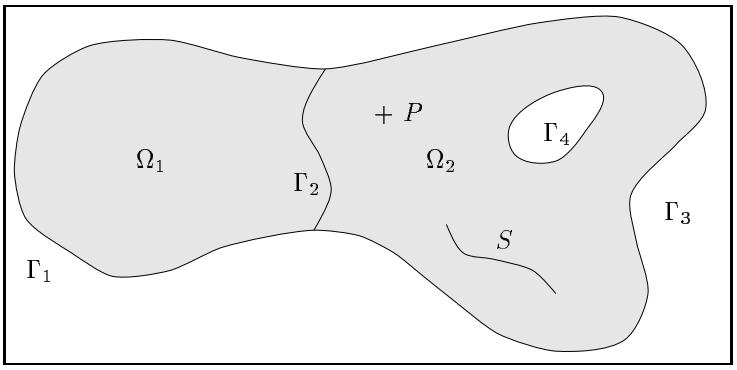



Figure 3.2: Example of a domain $\Omega = \Omega_1 \cup \Omega_2$.

Physical properties can be associated to some of the entities we have just recalled (sub-domains, boundaries, isolated segments and isolated points). For instance, in the case of a thermal problem, one can associate to a sub-domain its conductivity and its heat production, to a boundary its transfer coefficient and its external temperature, and to an isolated point its temperature.

### The splines

It is convenient to represent each boundary segment or isolated segment as one or several spline(s). Normally, one spline alone does not contain an angular point. On the other hand, the joining point between two consecutive splines may be angular (figure 3.3).

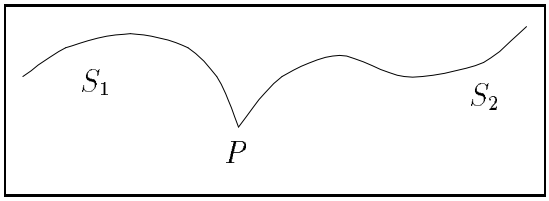



Figure 3.3: Curved segment made up of two splines $S_1$ and $S_2$.

Each spline is calculated in such a way that it passes through some given points. Two other control points may also be given if the user wishes to specify the tangent vector at each end. Many different methods have been studied to achieve this goal. In the BL2D software package, we have implemented three main methods, called "pure Catmull-Rom", "modified Catmull-Rom", and "elastic curve with minimum energy":

❶ A "pure Catmull-Rom" spline [4] is defined by a control polygon whose vertices are $\{P_i\}_{i=1..n}$. The spline passes through points $\{P_i\}_{i=2..n-1}$ and, at each point $P_i$, the tangent vectors to the two adjacent spline pieces $P_{i-1}P_i$ and $P_iP_{i+1}$ both equal $\frac{1}{2}\overrightarrow{P_{i-1}P_{i+1}}$, ensuring a $C^1$ continuity of the curve.

For instance (figure 3.4), the spline $P_2P_3P_4P_5$ is defined by the control polygon $P_1P_2P_3P_4P_5P_6$. The spline piece $P_2P_3$ is a cubic piece defined by its two ends $P_2$ and $P_3$, by the tangent at $P_2$ equal to $\overrightarrow{P_1P_3}/2$, and by the tangent at $P_3$ equal to $\overrightarrow{P_2P_4}/2$. The cubic pieces $P_3P_4$ and $P_4P_5$ are defined in a similar way.

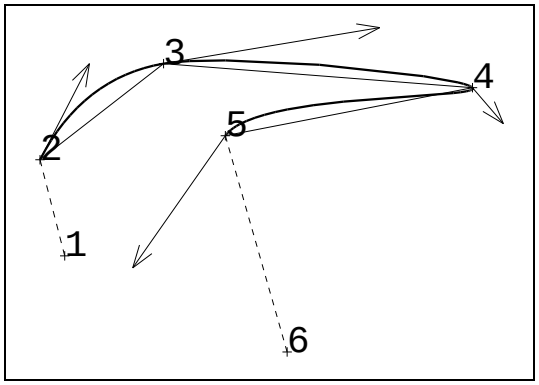



Figure 3.4: A spline and its control polygon.

Among the vertices of the control polygon $\{P_i\}_{i=1..n}$ ($n \geq 4$, $P_1$ and $P_n$ optional), we can distinguish:

- The two ends of the spline: $P_2$ and $P_{n-1}$.
- The control points that are internal to the spline: $P_3 \ldots P_{n-2}$. If $n = 4$, these points are absent. Otherwise ($n > 4$), the spline will pass through these points $\{P_i\}$. Let's recall that, at each point $P_i$, the tangent to the pieces $P_{i-1}P_i$ and $P_iP_{i+1}$ is continuous and equal to the vector $\overrightarrow{P_{i-1}P_{i+1}}/2$.
- The two control points that are external to the spline: $P_1$ and $P_n$. If the point $P_1$ (resp. $P_n$) is present, it is used to fix the tangent at the end point $P_2$ (resp. $P_{n-1}$) of the spline, using the same equality. If a point is absent, the tangent at the associated end point is considered as free.

So, each spline piece $P_2P_3$, ..., $P_{n-2}P_{n-1}$ is defined by its two ends and possibly the tangents at its two ends. If the tangents are both fixed, the piece is cubic (degree $\leq 3$). If only one tangent is fixed, the piece is parabolic (degree $\leq 2$). Finally, if no tangent is fixed, the piece is linear (degree $\leq 1$). As a consequence, if only the two ends of the spline are defined (i.e. neither internal nor external control points exist), the spline is reduced to a straight segment.

However, pure Catmull-Rom method may give bad results if the lengths of the edges of the control polygon are very different. For instance, the first spline in figure 3.5 has four control points ($P_1$ and $P_4$ are not represented in the figure): $P_1 = (1 - 5\sqrt{2}, -5\sqrt{2})$, $P_2 = (0, 0)$, $P_3 = (1, 0)$ and $P_4 = (5\sqrt{2}, -5\sqrt{2})$. The chord length is $|\overrightarrow{P_2P_3}| = 1$, and the norms of the tangent vectors are $|\overrightarrow{P_1P_3}/2| = |\overrightarrow{P_2P_4}/2| = 5$. This big ratio (that is equal to 5) gives here a loop which is usually undesirable. Similar problems are shown in the two other splines in figure 3.5. This would also occur in figure 3.8 (Ouragan airplane), though the cusps appearing in the dotted circles would be difficult to see directly. A last example with two loops is shown in figure 3.9.

❷ One way of overcoming these problems is to normalize the two tangent vectors to the chord length, and to make sure that the angle between the chord and the tangent is less than 45° (see figures 3.6 and 3.10). This gives a $G^1$ instead of a $C^1$ continuity between two adjacent pieces, but this is usually enough. This method has been implemented in the `blsmooth` program with the option "modified Catmull-Rom" (in fact two options are available, the second one being more general as the norm is multiplied by a given coefficient $\alpha$).

❸ A more elaborate way of overcoming these problems is to model the curve as an elastic wire [8]. If it is deformed by external constraints or forces, its shape is determined to make its stored elastic energy minimum. This gives smooth shapes with $G^2$ continuity (see figure 3.11). This

method has been implemented in the `blsmooth` program with the option "elastic curve" (in fact we have to solve a system of three-term simultaneous equations that is generally ill-conditioned, and two options are available: the first one is relatively safe, using a complete Gauss pivoting method; the second one is optimized in time and memory, but the solution may be swamped by finite-precision calculations if there are two many control points, say 30). Three examples using this method are shown in figures 3.7, 3.8 and 3.11.

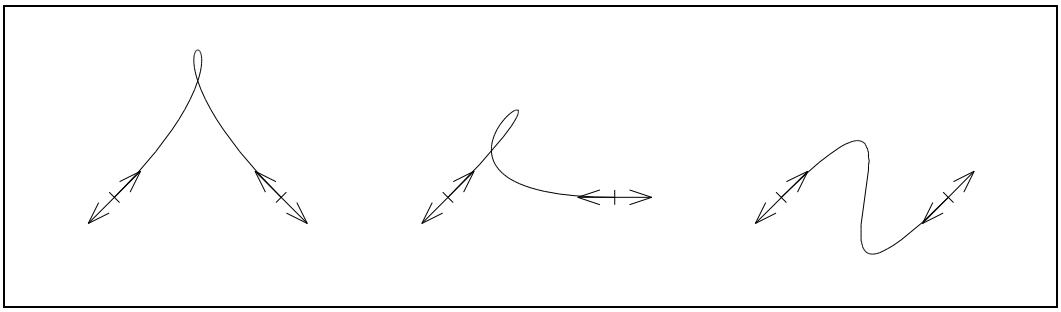


Figure 3.5: Pure Catmull-Rom: here $|\dot{r}(0)| = |\dot{r}(1)| = 5\,|chord|$.

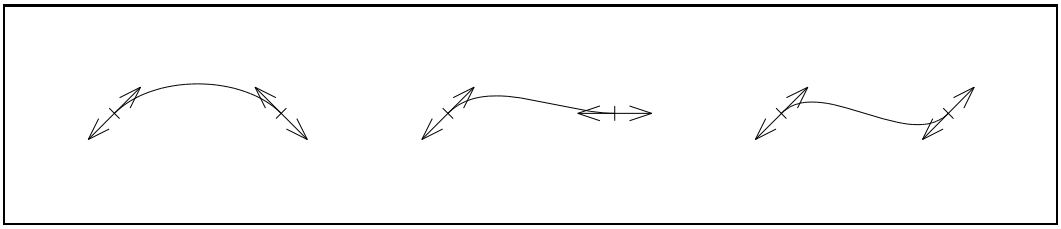


Figure 3.6: Modified Catmull-Rom: $|\dot{r}(0)| = |\dot{r}(1)| = |chord|$.

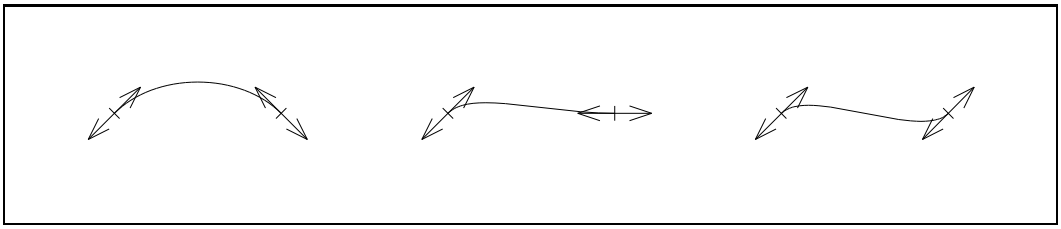


Figure 3.7: Elastic curve with minimum energy.

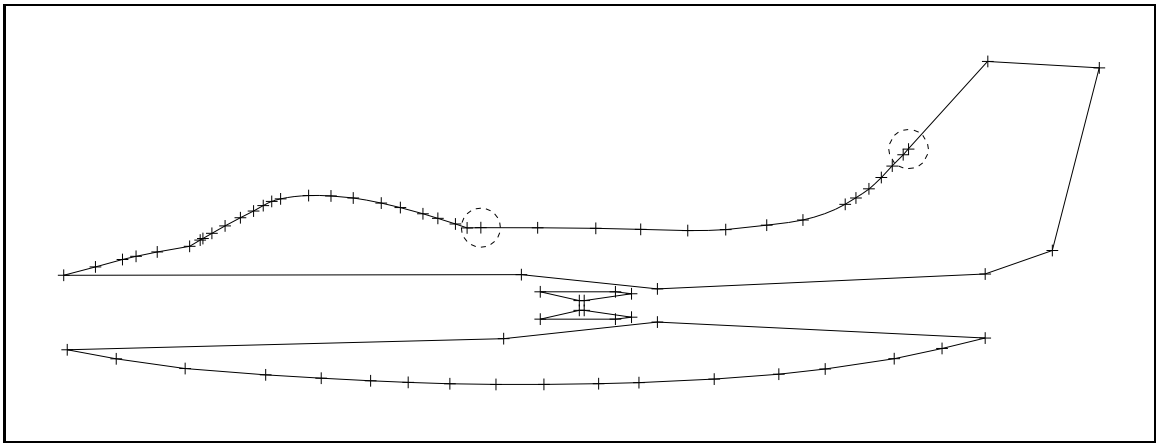


Figure 3.8: Ouragan airplane (elastic curve).

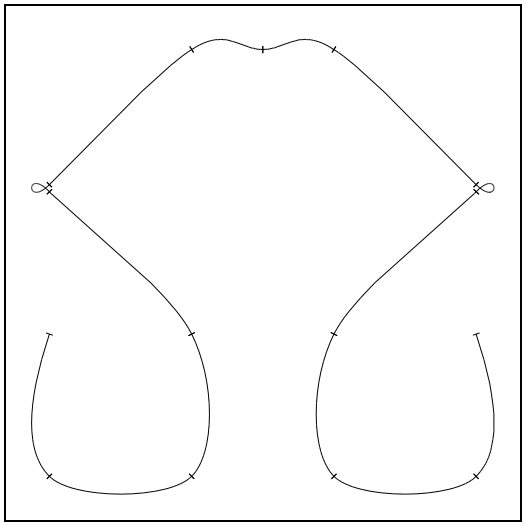


Figure 3.9: Pure Catmull-Rom.

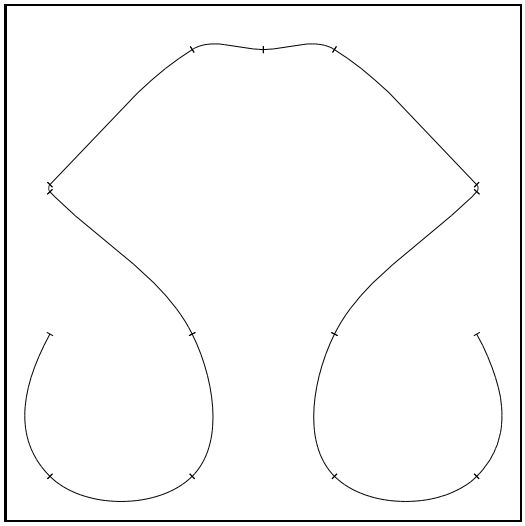


Figure 3.10: Modified Catmull-Rom.

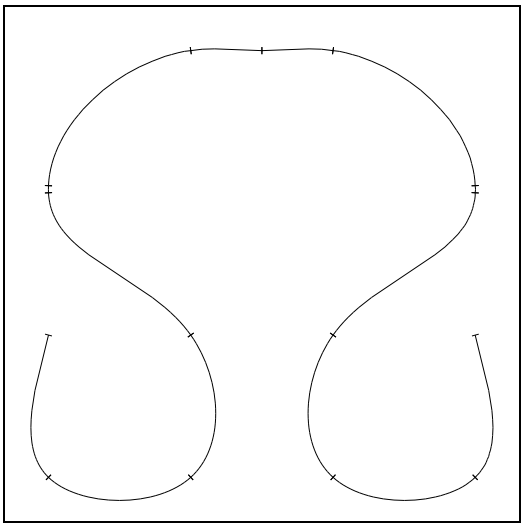


Figure 3.11: Elastic curve.

### Circles

A circle (or an arc) is a geometric form that is often encountered, for instance in the representation of various mechanical parts. In the BL2D software package, a circle is always approximated by a spline. Then, to reach a correct accuracy, it is necessary to specify a control polygon with a sufficient number of edges (of the same length). Let this number of edges (or vertices) be $n$. Note that the control polygon has $n+3$ control points $P_nP_1P_2P_3P_4 \ldots P_nP_1P_2$ with the first three and the last three vertices coinciding. To the naked eye, the number $n = 4$ is unacceptable for the "pure Catmull-Rom" method, but $n = 8$ is acceptable for the three possible methods (see figures 3.13 to 3.15). More precisely, the curves below give, for certain values of the number $n$, the maximum distance $\Delta$ between a circle of radius 1 and an approximating cubic spline.

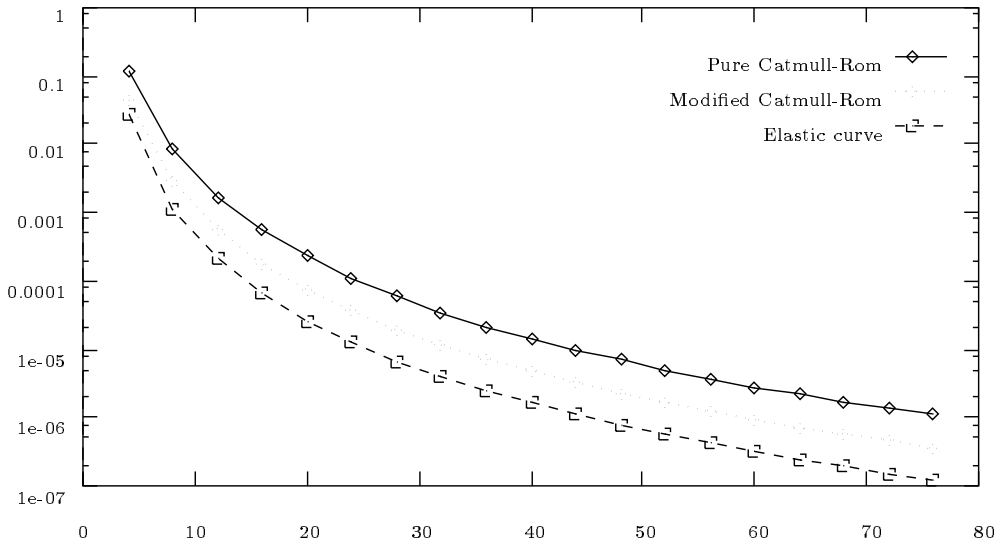



Figure 3.12: Error $\Delta$ (log scale) as a function of the number of vertices $n$.

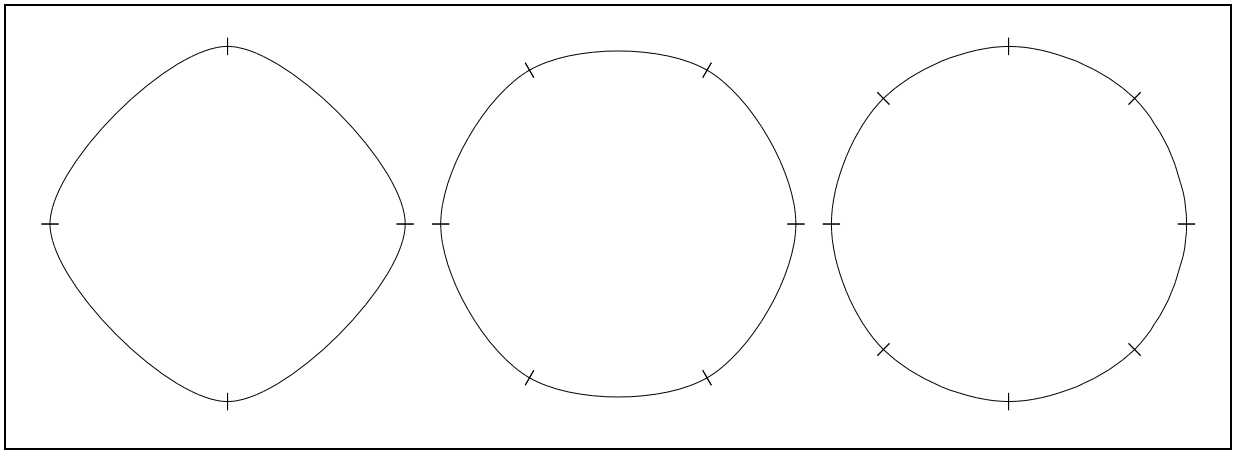


Figure 3.13: Pure Catmull-Rom.

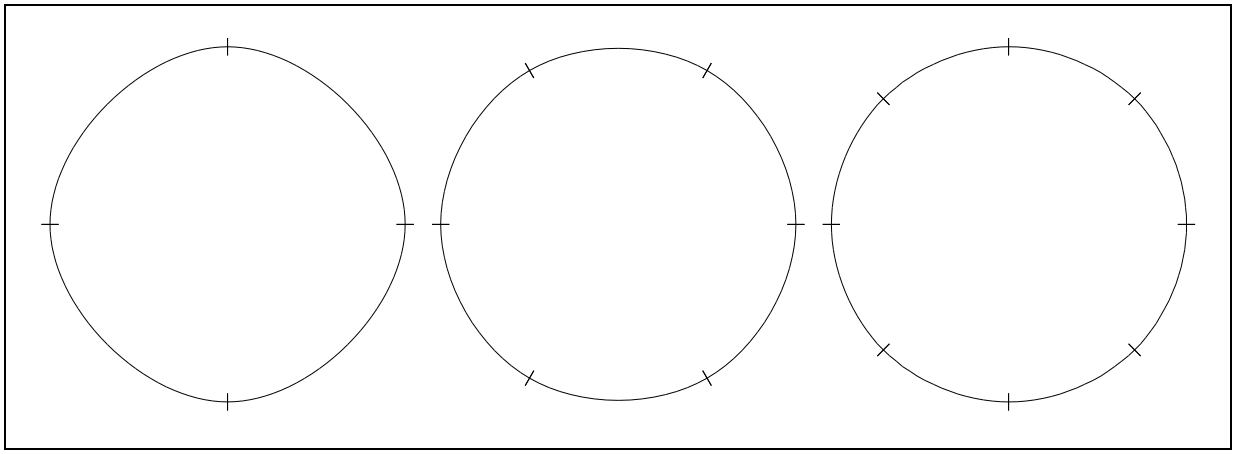


Figure 3.14: Modified Catmull-Rom.

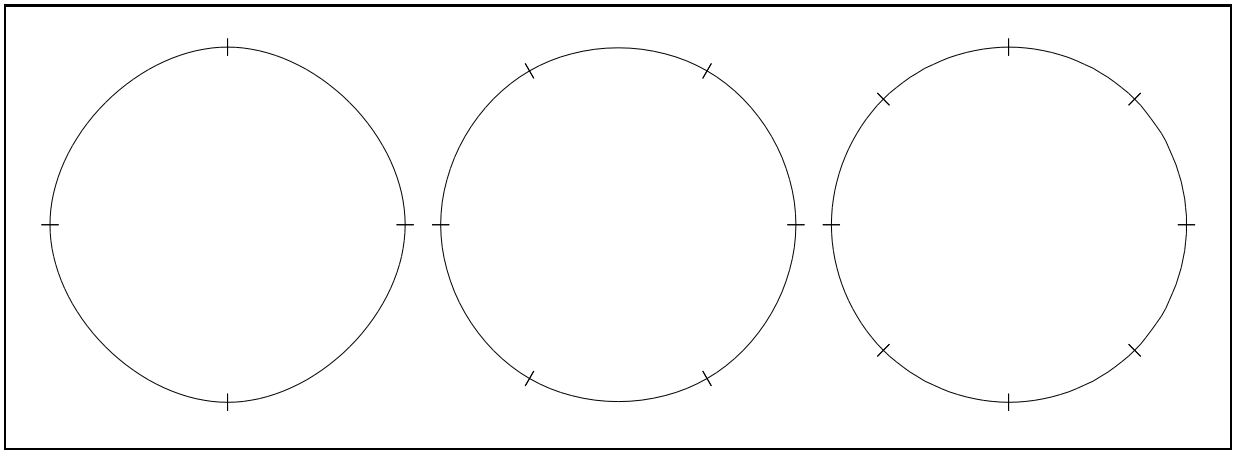


Figure 3.15: Elastic curve.

### Definition of the G format

`G v1`
NP NPI NPE NPG NS ND
For all the points ..... $i = 1..NP$
$x_i$ $y_i$
For all the isolated points ..... $i = 1..NPI$
$p_i$ $\varphi_i$
For all the end points ..... $i = 1..NPE$
$p_i$ $\varphi_i$
For all the splines ..... $i = 1..NS$
$np_i$
$(p_{ij})_{\ j=1..np_i}$ $\varphi_i$
For all the sub-domains ..... $i = 1..ND$
$s_i$ $o_i$ $\varphi_i$

### Notations

NP Total number of points: NP = NPI + NPE + NPG.
NPI Number of isolated points.
NPE Number of end points.
NPG Number of remaining points (i.e. neither isolated nor end points).
NS Number of splines.
ND Number of sub-domains.

For a point $i = 1..NP$:

$x_i$ $y_i$ Coordinates of the point.

For an isolated point, $i = 1..NPI$:

$p_i$ Number of the point.
$\varphi_i$ Physical reference of the point.

For an end point, $i = 1..NPE$:

$p_i$ Number of the point.
$\varphi_i$ Physical reference of the point.

For a spline $i = 1..NS$:

$np_i$ Number of points used for defining the spline ($np_i \geq 4$).
$p_{ij}$ Numbers of the points used for defining the spline, given in the following order:

- $p_{i,1}$ Number of the external control point 1. If this point is absent, $p_{i,1} = 0$.
- $p_{i,2}$ Number of the end point 1.
- $p_{i,3..np_i-2}$ Numbers of the control points that are internal to the spline, starting from end point 1 and going as far as end point 2.
- $p_{i,np_i-1}$ Number of the end point 2.
- $p_{i,np_i}$ Number of the external control point 2. If this point is absent, $p_{i,np_i} = 0$.

$\varphi_i$ Physical reference of the spline.

For a sub-domain $i = 1..ND$:

$s_i$ Number of a spline belonging to the boundary.

$o_i$ Direction so that the sub-domain is on the left of the spline $s_i$. If $o_i = 1$, we are starting from end point 1 and going as far as end point 2. If $o_i$ = -1, we are going in the opposite direction.

$\varphi_i$ Physical reference of the sub-domain.

**Remarks**

The first loop $i$ = 1..NP defines the coordinates of all the points used in the G format. These coordinates must be different each other.

A point is defined as:

- an isolated point if it appears in the loop $i$ = 1..NPI. The numbers $p_i$ of the points must be different each other.
- an end point if it appears in the loop $i$ = 1..NPE. Again, the numbers $p_i$ of the points must be different each other. Moreover, this loop must contain all the points that appear as end points in the description of the splines (points $p_{i,2}$ and $p_{i,np_{i-1}}$).
- an external control point if it appears as such in the description of the splines (points $p_{i,1}$ and $p_{i,np_i}$).
- an internal control point if it appears as such in the description of the splines (points $p_{i,3..np_{i-2}}$).
- a point with a metric if the HG file defines at this point a given size or metric ($h_i \neq 0$ or $a_i \neq 0$, see section 3.4).

The different possible combinations are specified in the symmetric table below (with the abbreviation c.p. = control point):

| — | isolated | end point | external c.p. | internal c.p. | with a metric |
|---|---|---|---|---|---|
| isolated | — | forbidden | possible | possible | obligatory |
| end point | | — | possible | forbidden | obligatory |
| external c.p. | | | — | possible | useless |
| internal c.p. | | | | — | possible |
| with a metric | | | | | — |

The combination "isolated - end point" is forbidden in order to number the points of the meshes in a systematic way (see sections 3.2 and 3.7).

The combination "isolated - external c.p." is useful in the definition of certain splines.

The combination "isolated - internal c.p." is useful to impose a point to the curve mesh generator without artificially dividing a spline.

The combination "end point - external c.p." is useful in the case of curves that are almost closed.

The combination "end point - internal c.p." is forbidden because two splines may be adjacent only if they have common end points.

The combination "external c.p. - internal c.p." is useful in the case of closed curves.

The combination "internal c.p. - point with a metric" is useful to govern more accurately the curve mesh generator.

In the description of a point, a spline or a sub-domain, we call physical reference $\varphi_i$ an integer number that is used as an index to the physical properties of this entity. By convention, $\varphi_i = 0$ means that no physical property must be associated to this entity.

### Example of a file in the G format

The G file below has been used in our example (see section 1.2).

```
G v1
9 1 5 3 5 1
  0.0000000000000000   0.0000000000000000
  1.0000000000000000   0.0000000000000000
  0.0000000000000000   1.0000000000000000
  0.4000000000000000   0.4000000000000000
  1.3535533905932737   0.6464466094067263
  1.0000000000000000   0.5000000000000000
  0.6464466094067263   0.6464466094067263
  0.5000000000000000   1.0000000000000000
  0.6464466094067263   1.3535533905932737

4 10

1 20
2 30
3 40
6 50
8 60

4
0 1 2 0 110
4
0 2 6 0 120
5
5 6 7 8 9 130
4
0 8 3 0 140
4
0 3 1 0 150

1 1 200
```

## 3.4 The H input format

### Purpose

A mesh generator must create curve elements or domain elements. The purpose of the H format is to govern the mesh generator by giving, in the neighborhood of certain points of a domain, the desired size and/or shape of the elements to create. The points of the domain are themselves defined in the G format (geometrical and physical data), the C format (curve mesh) or the MS format (domain mesh).

These data are usually generated by a preprocessor (to govern the initial mesh) or by an *estimator* (to adapt the mesh).

### Preliminary definitions

The map of sizes or metrics is defined on a set of points, that are themselves given in the G, C or MS format.

If the mesh to create is isotropic, it is sufficient to give at each point the desired size of the elements to generate in the neighborhood of this point.

If the mesh to create is anisotropic, the retained method consists in giving at each point a *metric* in which the desired size is equal to the unity [9]. A metric is represented by a symmetric positive

definite matrix with three coefficients $(a,\ b,\ c)$. Taking as the origin the point where the metric is given, any point $(x,\ y)$ located at a distance 1 from the origin, in the metric $(a,\ b,\ c)$, satisfies the equation

$$\begin{pmatrix} x & y \end{pmatrix} \begin{pmatrix} a & b \\ b & c \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = 1 \qquad \Longleftrightarrow \qquad a\,x^2 + 2\,b\,x\,y + c\,y^2 = 1.$$

This is the equation of an ellipse that is centered at the origin. By a rotation at an angle $\theta$ making a reference line parallel to one of the two axes of the ellipse (figure 3.16), the equation arises in the simplified form

$$\frac{X^2}{h_1^2} + \frac{Y^2}{h_2^2} = 1.$$

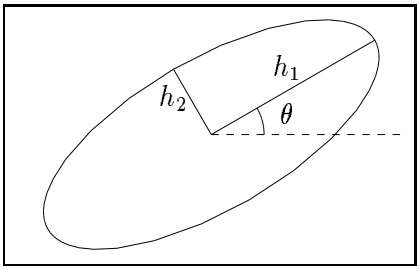



Figure 3.16: Ellipse defined by $\theta$, $h_1$ and $h_2$.

The values $h_1$ and $h_2$ represent the desired sizes along two orthogonal directions, in the usual metric equivalent to the identity. Conversely, if $\theta$, $h_1$ and $h_2$ are known, the metric $(a,\ b,\ c)$ is easily obtained thanks to the relation

$$\begin{pmatrix} a & b \\ b & c \end{pmatrix} = P \begin{pmatrix} \frac{1}{h_1^2} & 0 \\ 0 & \frac{1}{h_2^2} \end{pmatrix} P^{-1},$$

where $P$ is the "conversion matrix"

$$P = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \qquad \text{and} \qquad P^{-1} = \begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}.$$

## Definition of the H format

```
H v1
NP
isotropic   or   anisotropic
for each point ..... i = 1..NP
   if isotropic:    h_i
   if anisotropic:   a_i   b_i   c_i
```

### Notations

NP — Number of points where the sizes or the metrics are defined.

For a point $i = 1..NP$:

$h_i$ — Size at point $i$ (isotropic case).

$a_i\ b_i\ c_i$ — Metric at point $i$ (anisotropic case).

### Remarks

It is sometimes useful to let the size or the metric free at certain points. By convention, the size or the metric is considered as free at any point $p_i$ such that $h_i = 0$ (isotropic case) or $a_i = 0$ (anisotropic case).

The metrics must be given at least at the isolated points and at the end points.

### Example of a file in the H format

The H file below has been used in our example (see section 1.2).

```
H v1
9
isotrope
  0.1000000000000000
  0.1000000000000000
  0.1000000000000000
  0.1000000000000000
  0.0000000000000000
  0.1000000000000000
  0.1000000000000000
  0.1000000000000000
  0.0000000000000000
```

## 3.5 The IS auxiliary format

### Purpose

During an adaption loop, several meshes of a same domain are generated. The purpose of the IS format is to give the barycentric coordinates of vertices in a "new mesh" (*foreground mesh*) in relation to vertices in an "old mesh" (*background mesh*). These data are created with a view to interpolate the solution of a finite element computation.

### Definition of the IS format

`IS v1`
NP
For all the points in the new mesh ..... $i = 1..\mathrm{NP}$
$p_{i1}$ $p_{i2}$ $p_{i3}$ $\lambda_{i1}$ $\lambda_{i2}$

### Notations

NP Number of points in the new mesh.

For a point in the new mesh $i = 1..\mathrm{NP}$:

$p_{i1}$ $p_{i2}$ $p_{i3}$ Numbers of three points in the old mesh, or 0 (see remarks).

$\lambda_{i1}$ $\lambda_{i2}$ Barycentric Coordinates (see remarks).

### Remarks

Let $P_i$ be the point number $i$ (in the new mesh).

In general, ($p_{i1}$, $p_{i2}$, $p_{i3}$) are the numbers of the vertices ($P_{i1}$, $P_{i2}$, $P_{i3}$) of the triangle in the old mesh that encloses point $P_i$. Then $P_i$ is vectorially defined by

$$P_i = \lambda_{i1} P_{i1} + \lambda_{i2} P_{i2} + (1 - \lambda_{i1} - \lambda_{i2}) P_{i3}.$$

However, it may be that no enclosing triangle exists (for instance if a curved boundary is remeshed, or if a curved segment is defined outside the domain). In that case, only the two enclosing points in the curve mesh are considered. By convention, the number of the third point is zero. The barycentric coordinates are then calculated as a function of the curvilinear abscissas. For instance, if $p_{i3} = 0$, we have

$$\lambda_{i1} = \lambda \qquad \lambda_{i2} = 1 - \lambda \qquad 1 - \lambda_{i1} - \lambda_{i2} = 0.$$

### Example of a file in the IS format

The IS file below has been used in our example (see section 1.2).

```
IS v1
204
 49     1    65   0.000000   1.000000
  2    39     0   1.000000   0.000000
  3    16     0   1.000000   0.000000
  4    31     0   1.000000   0.000000
  5    20   101   1.000000   0.000000
  6   105    26   1.000000   0.000000
  7     8     0   0.672134   0.327866
  8     9    61   0.422772   0.577228
  9    10     0   0.242179   0.757821
...
 30     4    71   0.051587   0.172009
  3    16    48   0.087147   0.073293
  5   101    19   0.457181   0.475098
```

## 3.6 The MC auxiliary format

### Purpose

The purpose of the MC format is to describe all the geometric data that are necessary to the `blms` domain mesh generator. Therefore, it mainly describes a curve mesh. These data are generated by the `blprems` program from the G, SMOOTH and C formats.

### Definition of the C format

NP NA ND
$bb_1$ $bb_2$ $bb_3$ $bb_4$
For all the points ..... $i = 1..NP$
  $x_i$ $y_i$
For all the edges ..... $i = 1..NA$
  $p_{i1}$ $p_{i2}$
For all the sub-domains ..... $i = 1..ND$
  $q_{i1}$ $q_{i2}$

### Notations

| | |
|---|---|
| NP | Number of points. |
| NA | Number of edges. |
| ND | Number of sub-domains. |
| $bb_1..bb_4$ | Bounding box:<br>$(bb_1, bb_2)$ are the coordinates of the lower left corner,<br>$(bb_3, bb_4)$ are the coordinates of the upper right corner. |

For a point $i = 1..NP$:

$x_i$ $y_i$ Coordinates of the point.

For an edge $i = 1..NA$:

$p_{i1}$ $p_{i2}$ Numbers of the two end points of the edge.

For a sub-domain $i = 1..ND$:

$q_{i1}$ $q_{i2}$ Numbers of the two end points of an edge that is on the boundary or inside a sub-domain. The order of the end points is such that the sub-domain is on the left side when going from $q_{i1}$ to $q_{i2}$.

### Example of a file in the MC format

The MC file below has been used in our example (see section 1.2).

```
39 38 1
0.0000000000000000   0.0000000000000000
1.0000000000000000   1.0000000000000000

0.4000000000000000   0.4000000000000000
0.0000000000000000   0.0000000000000000
1.0000000000000000   0.0000000000000000
...
2 7
7 8
8 9
...
2 7
```

## 3.7 The MS output format

### Purpose

In a domain mesh, each sub-domain is subdivided into triangles (quadrangles are not available in the present release). The purpose of the MS format is to describe this set of triangles.

These data are generated by the `blms` program.

### Definition of the MS format

NP NT

For all the points ..... $i = 1..NP$

$x_i$ $y_i$

For all the triangles ..... $i = 1..NT$

$p_{i1}$ $p_{i2}$ $p_{i3}$ $v_{i1}$ $v_{i2}$ $v_{i3}$ $d_i$

### Notations

NP Number of points.

NT Number of triangles.

For a point $i = 1..NP$:

$x_i$ $y_i$ Coordinates of the point.

For a triangle $i = 1..NT$:

$p_{i1}$ $p_{i2}$ $p_{i3}$ Numbers of the 3 vertices, given counterclockwise.

$v_{i1}\ v_{i2}\ v_{i3}$ Numbers of the 3 connected triangles.

$d_i$ Number of the sub-domain:
$d_i = 0$ for the sub-domain that lies between the object and the *bounding box*,
$1 \leq d_i \leq$ ND for the sub-domains defined in G,
$d_i \geq$ ND $+ 1$ for the other sub-domains ("holes").

### Remarks

When the points are numbered (from 1 to NP), we consider successively:

- the NPI isolated points defined in G,
- the NPE end points defined in G,
- the NPC points created by the curve mesh generator (see section 3.2),
- the points created by the domain mesh generator,
- the 4 points of the *bounding box*.

The physical references of the points, edges and triangles, that are already described in the G format, are not duplicated in the MS format. They can be obtained by functions that are provided for this purpose (see section 5.2).

### Example of a file in the MS format

The MS file below has been used in our example (see section 1.2).

```
125 244
 3.99999985098829058e-01   3.99999985098829058e-01
-1.49011709638166786e-08  -1.49011709638166786e-08
 9.99999985098829036e-01  -1.49011709638166786e-08
-1.49011709638166786e-08   9.99999985098829036e-01
 9.99999985098829036e-01   4.99999985098829036e-01
...
 69    70    98    57   128   193   1
 96    92    93   109   235    95   1
125    10     9   160    37    20   0
125    35   124    46     0    42   0
122     5   123    30     0    64   0
...
```

## 3.8 The P auxiliary format

### Purpose

Let's recall that the `blprepro` program creates the initial files G (geometrical and physical data) and HG (map of sizes or metrics) (see section 1.1). The purpose of the P format is to allow the user to describe these data as simply as possible.

### General principles

The P format is like a catenation of the G and H formats, but offers the following capabilities:

- The geometric objects (points, splines, sub-domains, ...) can be identified by names instead of numbers.
- The number of objects are automatically calculated.

- The input data are read in a free format [5]. It is therefore possible to include comments and arithmetic expressions.
- The anisotropic metrics are given in the form $\theta$, $h_1$, $h_2$ instead of $a$, $b$, $c$. The angle $\theta$ is given in degrees. The value $h_1$ (resp. $h_2$) is the desired size along the axis inclined with the angle $\theta$ (resp. $\theta + 90°$).

### Definition of the P format

`P v1`
NP NS ND
For all the points $i$:
$id_i$ $x_i$ $y_i$
;
For all the isolated points:
$id_i$
;
For all the splines:
$id_i$ of the spline
$id.$ of the external control point 1 (`NULL` if it is absent)
$id.$ of the end point 1
$id.$ of the internal control points
$id.$ of the end point 2
$id.$ of the external control point 2 (`NULL` if it is absent)
;
;
For all the isolated points or end points such that $\varphi_i \neq 0$:
$id_i$ $\varphi_i$
;
For all the splines such that $\varphi_i \neq 0$:
$id_i$ $\varphi_i$
;
For all the sub-domains such that $\varphi_i \neq 0$:
$id_i$ of a spline $o_i$ (+1 or -1) $\varphi_i$
;
`isotropic` or `anisotropic`
For all the isolated points or end points:
$id_i$
if `isotropic`: $h$ at point $id_i$
if `anisotropic`: $\theta$ $h_1$ $h_2$ at point $id_i$
;

### Notations

| | |
|---|---|
| NP | Maximal number of points. |
| NS | Maximal number of splines. |
| ND | Maximal number of sub-domains. |
| $id_i$ | Identifier of an objet $i$ (point, spline, sub-domain, ...). |
| $x_i$ $y_i$ | Coordinates of point $i$. |
| $\varphi_i$ | Physical reference of a point, a spline or a sub-domain. |

### Example of a file in the P format

The P file below has been used in our example (see section 1.2).

```
P v1
100 -- maximal number of points
100 -- maximal number of segments
100 -- maximal number of sub-domains

-- points
!X=SQRT(2)/4
A 0 0     B 1 0       C 0 1     P 0.4 0.4     D 1+X 1-X
E 1 0.5   F 1-X 1-X   G 0.5 1   H 1-X 1+X     ;

-- required points
P ;

-- segments
AB NULL A B NULL ;
BE NULL B E NULL ;
EG D E F G H ;
GC NULL G C NULL ;
CA NULL C A NULL ;
;

-- physical references of certain required points or end points
P 10   A 20   B 30   C 40   E 50   G 60   ;

-- physical references of certain segments
AB 110   BE 120   EG 130   GC 140   CA 150   ;

-- sub-domains
AB +1 200 ;

-- map of sizes or metrics
isotropic
A 0.1   B 0.1   C 0.1   P 0.1   E 0.1   F 0.1   G 0.1   ;
```

## 3.9 The SMOOTH auxiliary format

### Purpose

The purpose of the SMOOTH format is to represent splines, in order to compute efficiently the coordinates of points belonging to these splines.

### Definition of the SMOOTH format

To represent a spline, two kinds of methods can be considered:

- to make a mesh of curves that is finer as much as the curvature of the spline is stronger,
- to make a coarser mesh of curves, but to associate some data with it (e.g. the coefficients of cubic polynomials).

The first method has been implemented in the BL2D software package. For efficiency reasons, the fine mesh is preceded by its *bounding box*.

### Example of a file in the SMOOTH format

The SMOOTH file below has been used in our example. (see section 1.2). The quarter circle is approximated by two cubic splines (figure 3.17).

```
SMOOTH v1
  0.0000000000000000   0.0000000000000000
  1.0000000000000000   1.0000000000000000
C v1
37 1 5 31 5
  0.4000000000000000   0.4000000000000000
  0.0000000000000000   0.0000000000000000
  1.0000000000000000   0.0000000000000000
  ...
```

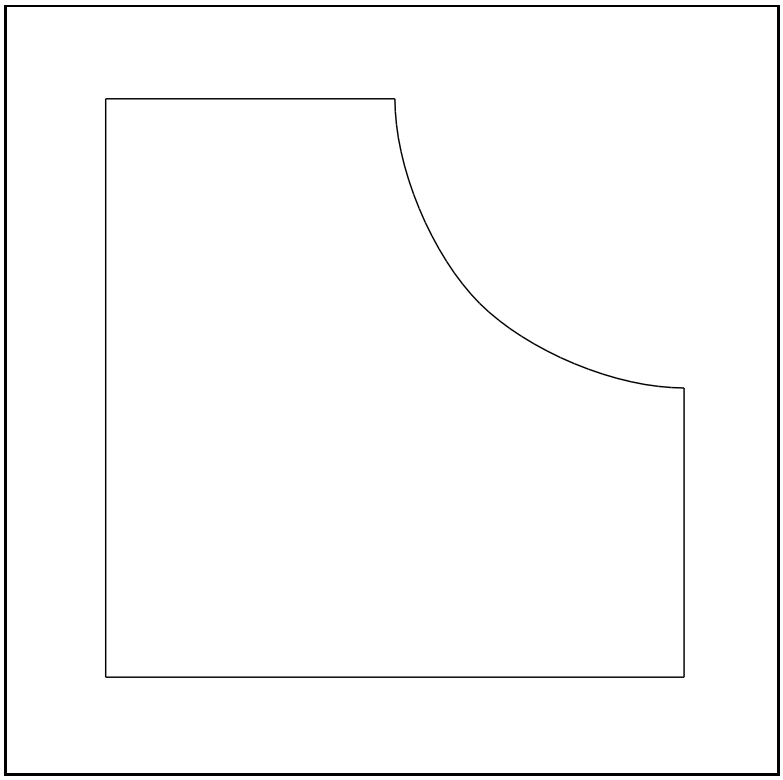


Figure 3.17: Illustration of the `quart.0.smooth` file.

# Chapter 4

# Data structures

The BL2D software package is written in two languages: Fortran 90 [10] and C. The programming of BL2D has been based on the concept of *class*, which gathers in a same entity *data structures* and *procedures*. The implementation in Fortran 90 is made according to the following correspondence table:

| "Object" terminology | Fortran 90 terminology |
|---|---|
| class | module |
| – structure | – derived type |
| – procedure | – procedure (subroutine or function) |

This chapter describes the main data structures implemented in Fortran 90. The procedures, as for themselves, are described in next chapter.

The previously defined formats can be represented in memory with data structures. So, to the G, C, MS and H formats correspond respectively the $g$, $c$, $s$ and $h$ structures. The organization of these structures being very close to that of formats, and the names of the components of the structures being relatively explicit, only the declarations in Fortran 90 are reproduced below.

## 4.1 The $g$ structure

```
type g_point_
   integer :: ip, phys
end type g_point_

type g_spline_
   integer, dimension(:), pointer :: points
   integer :: phys
end type g_spline_

type g_domain_
   integer :: is, orientation, phys
end type g_domain_

type g_
   integer :: np, npi, npe, npg, ns, nd
   double precision, dimension(:,:), pointer :: coor   ! (2,:)
   type(g_point_),   dimension(:), pointer :: iso_points
   type(g_point_),   dimension(:), pointer :: end_points
   type(g_spline_),  dimension(:), pointer :: splines
   type(g_domain_),  dimension(:), pointer :: domains
end type g_
```

## 4.2 The $c$ structure

```
type c_spline_
   integer, dimension(:), pointer :: points
   double precision, dimension(:), pointer :: abscissas
end type c_spline_

type c_
   integer :: np, npi, npe, npc, ns
   double precision, dimension(:,:), pointer :: coor   ! (2,:)
   type(c_spline_), dimension(:), pointer :: splines
end type c_
```

## 4.3 The $s$ structure

```
type s_
   integer :: np, nt
   double precision, dimension(:,:), pointer :: coor   ! (2,:)
   integer, dimension(:,:), pointer :: triangles       ! (3,:)
   integer, dimension(:,:), pointer :: neighbors       ! (3,:)
   integer, dimension(:),   pointer :: domains         ! (:)
end type s_
```

## 4.4 The $h$ structure

```
type h_
   integer :: np
   character(len=10) :: met_type
   double precision, dimension(:,:), pointer :: mets   ! (3,:)
end type h_
```

# Chapter 5

# Procedures

## 5.1 Reading and writing

The following procedures read or write structures of type $g$, $c$, $s$ or $h$ previously defined:

```
subroutine g_read(g)        subroutine g_write(g)
subroutine c_read(c)        subroutine c_write(c)
subroutine s_read(s)        subroutine s_write(s)
subroutine h_read(h)        subroutine h_write(h)
```

The extracts concerning array allocations, inside the read procedures, are listed below:

```
! structure g_
allocate(g%coor (1:2, g%np))
allocate(g%iso_points(g%npi))
allocate(g%end_points(g%npe))
allocate(g%splines   (g%ns))
allocate(g%domains   (g%nd))
do i = 1, c%ns
   allocate(g%splines(i)%points(n))
end do

! structure c_
allocate(c%splines(c%ns))
allocate(c%coor(1:2, c%np))
allocate(c%splines(c%ns))
do i = 1, c%ns
   allocate(c%splines(i)%points(n))
   allocate(c%splines(i)%abscissas(n))
end do

! structure s_
allocate(s%coor(1:2, s%np))
allocate(x_s%triangles(3, x_s%nt))
allocate(x_s%neighbors(3, x_s%nt))
allocate(x_s%domains  (   x_s%nt))

! structure h_
allocate(h%mets(3, h%np))
```

## 5.2 Physical references

The physical reference of a point `p`, an edge (`p`, `q`) or a triangle `t` is obtained respectively by the functions `phys_p`, `phys_a` and `phys_d`:

```
function phys_p(p, x_g, x_c) ! p = number in the mesh
  integer, intent(in) :: p
  type(g_), intent(in) :: x_g
  type(c_), intent(in) :: x_c
  integer :: phys_p ! intent(out)
end function phys_p

function phys_a(p, q, x_g, x_c)
  integer, intent(in) :: p, q
  type(g_), intent(in) :: x_g
  type(c_), intent(in) :: x_c
  integer :: phys_a ! intent(out)
end function phys_a

function phys_d(t, x_g, x_s)
  integer, intent(in) :: t
  type(g_), intent(in) :: x_g
  type(g_), intent(in) :: x_s
  integer :: phys_d ! intent(out)
end function phys_d
```

For instance, the three previous functions are called to draw the figures 5.1 to 5.3.

## 5.3 Splines

The function `eval_s` returns the coordinates of the point located on the spline `is` and whose curvilinear abscissa is `s`:

```
function eval_s(x_c, is, abscissa)
  type(c_), intent(in) :: x_c
  integer, intent(in) :: is
  double precision, intent(in) :: abscissa
  double precision :: eval_s(2) ! intent(out)
end function eval_s
```

Given an edge (`p`, `q`), the following subroutine checks if this edge belongs to a spline. If the answer is yes, it returns the number of the spline and the indices of the points `p` and `q`. Otherwise, it returns 0.

```
subroutine find_spline(p, q, x_c, is, ip, iq)
  integer, intent(in) :: p, q
  type(c_), intent(in) :: x_c
  integer, intent(out) :: is, ip, iq
end subroutine find_spline
```

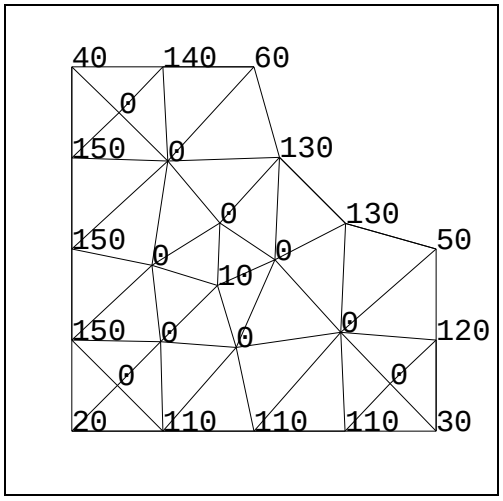



Figure 5.1: Physical references of the points.

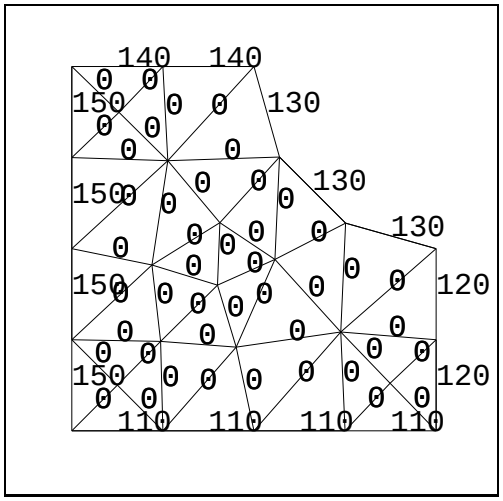



Figure 5.2: Physical references of the edges.

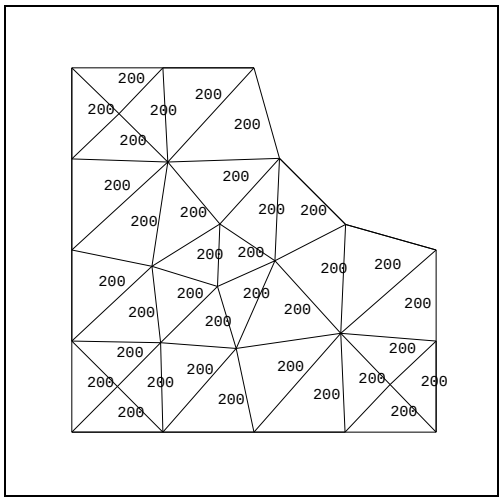



Figure 5.3: Physical references of the sub-domains.

By using the two previous procedures, one can for instance calculate the coordinates of the P2 node located at the middle of an edge (p, q):

```
call find_spline(p, q, x_c, is, ip, iq)
if (is == 0) then   ! straight edge
   coor = (x_s%coor(1:2,p) + x_s%coor(1:2,q)) / 2
else                ! curved edge
   abscissa = (x_c%splines(is)%abscissas(ip) + &
               x_c%splines(is)%abscissas(iq)) / 2
   coor = eval_s(x_smooth%c, is, abscissa)
end if
```

This method was applied to our preceding example. The nodes are correctly located on the quarter circle (figure 5.4).

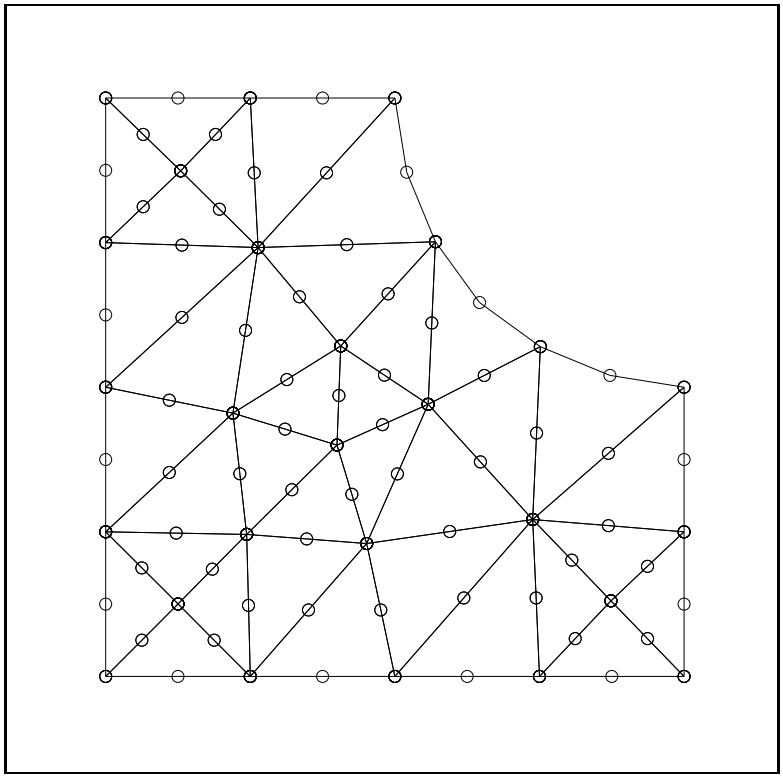


Figure 5.4: Triangles with P2 nodes.

# Bibliography

[1] H. Borouchaki, P. Laug, *Le mailleur adaptatif bidimensionnel BL2D : manuel d'utilisation et documentation*, Rapport Technique INRIA RT-0185, décembre 1995.

[2] H. Borouchaki, P.L. George, F. Hecht, P. Laug, E. Saltel, *Mailleur bidimensionnel de Delaunay gouverné par une carte de métriques. Partie I : Algorithmes*, Rapport de Recherche INRIA RR-2741, décembre 1995.

[3] H. Borouchaki, P.L. George, F. Hecht, P. Laug, B. Mohammadi, E. Saltel, *Mailleur bidimensionnel de Delaunay gouverné par une carte de métriques. Partie II : Applications*, Rapport de Recherche INRIA RR-2760, décembre 1995.

[4] E. Catmull, R. Rom, *A class of interpolating splines*, in *Computer Aided Geometric Design*, pp. 317-326, Academic Press, 1974.

[5] P.L. George, P. Laug, *Programming and Utilization*, Modulef User Guide n° 2, INRIA, 1992.

[6] P.L. George, E. Saltel, *Postprocessing and Graphics*, Modulef User Guide n° 6, INRIA, 1992.

[7] P. Laug, *Rapport de fin de phase 1 de "GÉNIE" – Tâches T-2.4.3 à T-2.4.6*, DASSAULT AVIATION - INRIA, juin 1996.

[8] M. Hosaka, *Modeling of Curves and Surfaces in CAD/CAM*, Computer Graphics – Systems and Applications, Springer-Verlag, 1992.

[9] P. Laug, H. Borouchaki, P.L. George, *Maillage de courbes gouverné par une carte de métriques*, Rapport de Recherche INRIA RR-2818, mars 1996.

[10] M. Metcalf, J. Reid, *Fortran 90 Explained*, Oxford University Press, 1990.

[11] E. Saltel, F. Hecht, *EMC$^2$ Wysiwyg 2D Finite Elements Mesh Generator*, available from ftp://ftp.inria.fr/INRIA/Projects/Gamma/emc2.doc.en.ps.gz, translated from *EMC$^2$ : un logiciel d'édition de maillages et de contours bidimensionnels*, Rapport Technique INRIA RT-0118, octobre 1995.

Unit´e de recherche INRIA Lorraine, Technopôle de Nancy-Brabois, Campus scientifique,
615 rue du Jardin Botanique, BP 101, 54600 VILLERS LÈS NANCY
Unit´e de recherche INRIA Rennes, Irisa, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unit´e de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unit´e de recherche INRIA Rocquencourt, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex
Unit´e de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 SOPHIA-ANTIPOLIS Cedex

Éditeur
INRIA, Domaine de Voluceau, Rocquencourt, BP 105, 78153 LE CHESNAY Cedex (France)
ISSN 0249-6399